Process/Thread Pinning Overview

Category: Process Pinning

Columbia Phase Out:

As of Feb. 27, 2013, the Columbia21, Columbia23, and Columbia24 nodes have been taken offline as part of the Columbia phase out process. Columbia22 is still available. If your script requires a specific node, please make the appropriate changes in order to ensure the success of your job.

Summary: Pinning, the binding of a process or thread to a specific core, can improve the performance of your code by increasing the percentage of local memory accesses.

Once your code runs and produces correct results on a system, the next concern is its performance. For a code that uses multiple cores, the placement of processes and/or threads can play a significant role in code performance.

Given a set of processor cores in a PBS job, the Linux kernel usually does a reasonably good job of mapping processes/threads to physical cores (although the kernel may also migrate processes/threads). Some OpenMP runtime libraries and MPI libraries may also perform certain placements by default. In cases where the placements by the kernel or the MPI or OpenMP libraries are suboptimal, you can try multiple methods to control the placement in order to improve performance of your code. Using the same placement also has the added benefit of reducing runtime variability from run to run.

You should pay attention to maximizing data locality while minimizing latency and resource contention, and should have a clear understanding of the characteristics of your own code and the machine that the code is running on.

Characteristics of NAS HECC Systems

Pleiades and Columbia are two distinctly different types of systems.

Pleiades

Pleiades is a cluster system consisting of four different processor types -- Harpertown, Nehalem, Westmere, and Sandy Bridge, with a total of 11,776 nodes. On Pleiades, memory on each node is accessible and shared only by processes/threads running on that node.

A Harpertown node is a symmetric memory system where all 8 cores have equal access to the memory on the node, so data locality is not an issue.

On the other hand, a Nehalem-EP, Westmere, or Sandy Bridge node contains two sockets. Within each socket is a symmetric memory system. Accessing memory across the two sockets is through the Quick Path Interconnect and these nodes are considered non-uniform memory access (NUMA) systems. So,for optimal performance, data locality should not be overlooked on these three processor types.

Overall, compared to a global shared-memory NUMA system such as Columbia, data locality is less of a concern on Pleiades. Rather, minimizing latency and resource contention will be the main focus when pinning processes/threads on Pleiades.

For more information on Pleiades and these processors, see Pleiades Configuration Details, which has links to each of the processor types.

Columbia

Columbia comprises 4 hosts (C*21-24*). Each host is a NUMA system that contains hundreds of nodes with memory located physically at various distances from the processors accessing data on memory. A process/thread can access the local memory on its node, as well as the remote memory across nodes through the NUMAlink, with varying latencies. So, data locality is critical for getting good performance on Columbia.

One good practice to follow when developing an application is to initialize data in parallel, such that each processor core initializes data that it is likely to access later for calculation.

For more information about Columbia, see Columbia Configuration Details.

Methods for Process/Thread Pinning

Several pinning approaches for OpenMP, MPI and MPI+OpenMP hybrid applications are listed below. We recommend using the Intel compiler (and its runtime library) and the SGI MPT software on NAS systems, so most of the approaches pertain specifically to them. On the other hand, the **`mbind`** tool works for multiple OpenMP libraries and MPI environments.

- OpenMP codes
    - ♦ Using Intel OpenMP Thread Affinity for Pinning
    - ♦ Using SGI's omplace Tool for Pinning
    - ♦ Using the mbind Tool for Pinning
- MPI codes

- Setting SGI MPT Environment Variables

- Using SGI's omplace Tool for Pinniing
- Using the mbind Tool for Pinning

- MPI+OpenMP hybrid codes

- Using SGI's omplace Tool for Pinning
- Using the mbind Tool for Pinning

Checking Process/Thread Placement

Each of the approaches listed above provides some verbose capability to print out the tool's placement results. In addition, you can check the placement using the following approaches:

ps Command

`ps -C` *`executable_name`* `-L -opsr,comm,time,pid,ppid,lwp`
In the output generated, use the core ID under the PSR column, the process ID under the PID column, and the thread ID under the LWP column to see where the processes and/or threads are placed on the cores.

Note that the **`ps`** command provides a snapshot of the placement at that specific time. You may need to monitor the placement from time to time to make sure that the processes/threads do not migrate.

Instrument your code to

- Call the **`mpi_get_processor_name`** function, to get the name of the processor an MPI process is running on
- Call the Linux C function **`sched_getcpu()`** to get the processor number the process or thread is running on

For more information, see Instrumenting your Fortran Code to Check Process/Thread Placement.

Article ID: 259
Last updated: 14 Feb, 2013
Computing at NAS -> Best Practices -> Process Pinning -> Process/Thread Pinning Overview

http://www.nas.nasa.gov/hecc/support/kb/entry/259/?ajax=1